Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 185 (2007) 93–105
www.elsevier.com/locate/entcs

Formal Verification of Concurrent Systems via Directed Model Checking

Sara Gradara, Antonella Santone and Maria Luisa Villani

RCOST - Research Centre on Software Technology
University of Sannio, Benevento, Italy
e-mail: {gradara,santone,villani}@unisannio.it

Abstract

Model checking suffers from the state explosion problem, due to the exponential increase in the size of a finite state model as the number of system components grows. Directed model checking aims at reducing this problem through heuristic-based search strategies. The model of the system is built while checking the formula and this construction is guided by some heuristic function. In this line, we have defined a structure-based heuristic function operating on processes described in the Calculus of Communicating Systems (CCS), which accounts for the structure of the formula to be verified, expressed in the selective Hennessy-Milner logic. We have implemented a tool to evaluate the method and verified a sample of well known CCS processes with respect to some formulae, the results of which are reported and commented.

Keywords: model checking, heuristic search, CCS, logic.

1 Introduction

Model checking [9] is a method to formally verify finite state concurrent systems. It suffers from the well-known state explosion problem caused by representing concurrency by interleaving. Not surprisingly, most of the research in model checking is focused on ways to minimize the impact of state explosion, as for example symbolic model checking [25], on-the-fly [23], local model checking [33], partial order [15,28], abstraction [8] and compositional reasoning [10,29]. Recently, great interest was shown in combining the two areas: model checking and heuristics to guide the exploration of the state graph of a system. For software validation, in the work of Yang and Dill [34] the bug-finding capability of a model checker is enhanced by using heuristics to search the states that are most likely to lead to an error. In [14] the concept of *directed model checking* has been introduced. Traditional model checking algorithms perform an uninformed state space exploration based on depth-first or breadth-first search algorithms. In directed model checking, heuristic search algorithms, such as A* [27], are used in order to guide the search to the shortest, or close to the shortest, path into a property-violating state. Several approaches

1571-0661 © 2007 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2007.05.031

have been proposed in the area of heuristic search for explicit state model checking, see for example [13,16,24]. Our work differs from those in that we provide an approach where heuristic searches are used to accelerate verification rather than finding errors.

In this paper we present our contribution to the verification of concurrent systems described as Calculus of Communicating Systems (CCS) [26] processes. We use the Selective Hennessy-Milner (SHM) [5] logic to express the properties to be verified. Following the classification in [32], the properties we manage successfully are weak liveness and safety. The idea of our approach is to avoid, if possible, the generation of the whole system's global state graph for verification. We use an heuristic function to prune the state space and to guide the state expansion towards the closest interesting states for the SHM formula at hand. The SHM formula to be verified is reduced along the state expansion according to some rules and it is taken into account for the generation of the transitions. We stop the construction of the state space as soon as we can deduce whether the original SHM formula is or not satisfied by the system.

If the heuristic function is consistent [1], the optimality of the solution is guaranteed when applying a search algorithm such as A*. In this paper we present one heuristic definition and examples that prove that this approach can really improve the search compared to exhaustive searches. Other (consistent) heuristic definitions at different levels of accuracy are presented in [20]. Our heuristic functions are syntactically defined, i.e., based on the CCS specifications and the structure of the formula. Also, they can be automatically computed, thus user intervention and manual efforts are not needed. A tool implementing our approach has been developed and real-life benchmark case studies have been considered for evaluation purposes.

2 Background

We assume that the reader is familiar with the basic concepts of heuristic searches and we refer to [27] for details.

2.1 The Calculus of Communicating Systems

We briefly recall the basic concepts about the Calculus of Communicating Systems (CCS) [26]. The syntax of *processes* is:

$$p ::= nil \mid \alpha.p \mid p + p \mid \ p|p \ \mid p\backslash L \mid p[f] \mid x$$

where α ranges over a finite set of actions $\mathcal{A} = \{\tau, a, \overline{a}, b, \overline{b}, ...\}$, τ is called the *internal action*. The set of *visible actions*, $\mathcal{V}$, is defined as $\mathcal{A} - \{\tau\}$, $L \subseteq \mathcal{V}$ and the relabeling function f is a total function, $f : \mathcal{A} \to \mathcal{A}$. Given $L \subseteq \mathcal{V}$, with $\overline{L}$ we denote the set $\{\overline{l} \mid l \in L\}$. x ranges over a set of *constant* names: each constant x

[1] A heuristic function is *consistent* if it the difference in the heuristic estimate between one state and its descendant is less than or equal to the actual path cost on the edge connecting them.

is defined by a constant definition $x \stackrel{\text{def}}{=} p$. The semantics of a process p is precisely defined by means of the structural operational semantics shown in Figure 1. The semantic definition describes the transition relation of the automaton corresponding to a CCS process p, called *standard transition system* for p, and denoted by $\mathcal{S}(p)$.

Act	$\dfrac{}{\alpha.p \xrightarrow{\alpha} p}$	**Sum**	$\dfrac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'}$ and sym.	**Par**	$\dfrac{p \xrightarrow{\alpha} p'}{p \mid q \xrightarrow{\alpha} p' \mid q}$ and sym.
Com	$\dfrac{p \xrightarrow{l} p',\ q \xrightarrow{\overline{l}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'}$	**Con**	$\dfrac{p_x \xrightarrow{\alpha} p'}{x \xrightarrow{\alpha} p'}\ x \stackrel{\text{def}}{=} p_x$	**Res**	$\dfrac{p \xrightarrow{\alpha} p'}{p\backslash L \xrightarrow{\alpha} p'\backslash L}\ \alpha \notin (L \cup \overline{L})$
		Rel	$\dfrac{p \xrightarrow{\alpha} p'}{p[f] \xrightarrow{f(\alpha)} p'[f]}$		

Fig. 1. Standard operational semantics of CCS

Given a process p, we use $\mathcal{F}irst(p) = \{\alpha | p \xrightarrow{\alpha} p'\}$ to denote the set of all the first actions that p can perform. It can be syntactically defined as the definition of sort given in [26].

Given a process p, a constant x of p is said to be *guarded in* p if x is contained in a sub-process of p of the form $\alpha.q$, where q is a process. A process p is *guarded* if every constant of p is guarded in p, it is *unguarded* otherwise. In the following, $Unfold^x(p)$ is the process obtained by replacing each unguarded constant x by its definition. For example, if $x \stackrel{\text{def}}{=} \overline{a}.x$, $Unfold^x((a.b.x|x)\backslash\{a\})$ is the process $(a.b.x|\overline{a}.x)\backslash\{a\}$.

From now on, for each process $q = (q_1|\cdots|q_n)$ we assume that if an action α belongs to the sort [2] of q_i, with $i \in [1..n]$ and $\overline{\alpha}$ belongs to the sort of q_j with $j \in [1..n]$ and $i \neq j$, then the process q occurs under a restriction set L such that $L \cup \overline{L}$ contains α. If both α and $\overline{\alpha}$ appear in a process, it is reasonable to assume that they are communication actions.

2.2 Selective Hennessy-Milner logic

The selective Hennessy-Milner logic (SHM logic) is a sub-logic of the selective mu-calculus, introduced by the author and others in [5], which is more expressive than the Hennessy-Milner logic [32]. The syntax of the SHM logic is the following, where K and R range over sets of actions:

$$\varphi ::= \texttt{tt} \mid \texttt{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid [K]_R\,\varphi \mid \langle K \rangle_R\,\varphi$$

The satisfaction of a SHM formula φ by a state s of a transition system, written $s \models \varphi$, is so defined:

- each state satisfies `tt` and no state satisfies `ff`;
- a state satisfies $\varphi_1 \vee (\wedge)\ \varphi_2$ if it satisfies φ_1 or (and) φ_2;

[2] The sort of a CCS process p is the alphabet of p [26].

- $[K]_R\,\varphi$ is satisfied by a state which evolves to a state obeying to φ, for every performance of a sequence of actions not belonging to $R \cup K$, followed by an action in K.
- $\langle K\rangle_R\,\varphi$ is satisfied by a state which can evolve to a state obeying to φ by performing a sequence of actions not belonging to $R \cup K$, followed by an action in K.

The selective modal operators $\langle K\rangle_R\,\varphi$ and $[K]_R\,\varphi$ substitute the standard modal operators $\langle K\rangle\,\varphi$ and $[K]\,\varphi$. We use this logic since only the actions explicitly mentioned by the selective modal operators can be used by the heuristic function suggesting the more promising nodes. We give some examples of SHM formulae to explain the use of the selective operators.

$\varphi_1 = [b]_{\{a\}}\,\mathtt{ff}$: "it is not possible to perform an action b if an action a has not been previously performed".
$\varphi_2 = [a]_{\{b\}}\,\langle c\rangle_{\emptyset}\,\mathtt{tt}$: "for each action a not preceded by an action b, it is possible to perform an action c preceded by any action".
$\varphi_3 = \langle a\rangle_{\{c\}}\,\mathtt{tt}$: "it is possible to perform an action a not preceded by an action c".

For example, the process: $p = a.b.c.nil + c.a.b.nil$ satisfies φ_1 and φ_3 while it does not satisfy φ_2.

3 A heuristic function for model checking CCS processes

The idea of the approach is to avoid the construction of the whole transition system when verifying the formulae, considering only the part of it that is sufficient to draw the right conclusions. For this purpose, we use a heuristic function suggesting which states to visit in order to be able to establish if the formula is satisfied or not as soon as possible during the construction of the state space. In our heuristic, the first actions occurring in the formula suggest the most promising nodes to expand.

Given a CCS process p and a SHM formula φ we formalize the formula verification as a state space search problem, as a quadruple $(\mathcal{N}, \mathcal{O}, N_0, \mathcal{G})$ where $\mathcal{N}$ is the set of nodes, i.e. a tuple $\langle s, \psi\rangle$ with s is a state belonging to the transition system $\mathcal{S}(p)$ and ψ is a sub-formula of φ; $\mathcal{O}$ is a set of operators $\mathcal{N} \longrightarrow \mathcal{N}$ defined in Figure 2; $N_0 = \langle p, \varphi\rangle$ is the initial node; and $\mathcal{G}$ is a subset of $\mathcal{N}$, called *goal nodes*. In the following we will define when a node is a goal node for the formula verification problem.

We informally explain the rules in Figure 2. In the **diamond**$_1$ rule, if p can perform α to become p', where $\alpha \in K$, i.e., α is the first action of φ, then the tuple $\langle p, \langle K\rangle_R\,\varphi\rangle$ can perform α to become $\langle p', \varphi\rangle$ where the formula is replaced with the remaining part after the first action α. If $\alpha \notin K \cup R$, as in the **diamond**$_2$ rule, then the tuple $\langle p, \langle K\rangle_R\,\varphi\rangle$ can perform α to become $\langle p', \langle K\rangle_R\,\varphi\rangle$ where the formula has not changed. This is the same for the **box**$_1$ and **box**$_2$ rules. It is worth noting that in the **diamond** rules the case of $\alpha \in R$ has not been included. This is because if

$$\textbf{diamond}_1 \quad \frac{p \xrightarrow{\alpha} p'}{\langle p, \langle K\rangle_R\, \varphi\rangle \xrightarrow{\alpha} \langle p', \varphi\rangle}\ \alpha \in K$$

$$\textbf{diamond}_2 \quad \frac{p \xrightarrow{\alpha} p'}{\langle p, \langle K\rangle_R\, \varphi\rangle \xrightarrow{\alpha} \langle p', \langle K\rangle_R\, \varphi\rangle}\ \alpha \notin R \cup K$$

$$\textbf{box}_1 \quad \frac{p \xrightarrow{\alpha} p'}{\langle p, [K]_R\, \varphi\rangle \xrightarrow{\alpha} \langle p', \varphi\rangle}\ \alpha \in K$$

$$\textbf{box}_2 \quad \frac{p \xrightarrow{\alpha} p'}{\langle p, [K]_R\, \varphi\rangle \xrightarrow{\alpha} \langle p', [K]_R\, \varphi\rangle}\ \alpha \notin R \cup K$$

$$\textbf{or}_1 \quad \frac{\langle p, \varphi_1\rangle \xrightarrow{\alpha} \langle p', \varphi_1'\rangle}{\langle p, \varphi_1 \vee \varphi_2\rangle \xrightarrow{\alpha} \langle p', \varphi_1'\rangle} \qquad \textbf{or}_2 \quad \frac{\langle p, \varphi_2\rangle \xrightarrow{\alpha} \langle p', \varphi_2'\rangle}{\langle p, \varphi_1 \vee \varphi_2\rangle \xrightarrow{\alpha} \langle p', \varphi_2'\rangle}$$

$$\textbf{and}_1 \quad \frac{\langle p, \varphi_1\rangle \xrightarrow{\alpha} \langle p', \varphi_1'\rangle}{\langle p, \varphi_1 \wedge \varphi_2\rangle \xrightarrow{\alpha} \langle p', \varphi_1'\rangle} \qquad \textbf{and}_2 \quad \frac{\langle p, \varphi_2\rangle \xrightarrow{\alpha} \langle p', \varphi_2'\rangle}{\langle p, \varphi_1 \wedge \varphi_2\rangle \xrightarrow{\alpha} \langle p', \varphi_2'\rangle}$$

Fig. 2. Operational semantics of $\langle p, \varphi\rangle$.

p could perform an action belonging to R that transition would not influence the truth-value of the formula. The case of $\alpha \in R$ has not been included in the **box** rule too, since in this case that transition of p always satisfies the formula $[K]_R\, \varphi$. The **or** (resp. **and**) rule, on the state $\langle p, \varphi_1 \vee \varphi_2\rangle$ (resp. $\langle p, \varphi_1 \wedge \varphi_2\rangle$) generates all the transitions for $\langle p, \varphi_1\rangle$ and $\langle p, \varphi_2\rangle$. The difference between them is in the definition of a goal for the formula verification. For example, consider the process $p = a.nil$ and the formulae $\psi_1 = \langle a\rangle_\emptyset\, \mathtt{tt} \vee \langle b\rangle_\emptyset\, \mathtt{tt}$ and $\psi_2 = \langle a\rangle_\emptyset\, \mathtt{tt} \wedge \langle b\rangle_\emptyset\, \mathtt{tt}$. Both $\langle p, \psi_1\rangle$ and $\langle p, \psi_2\rangle$ give rise to the following two transitions: $\xrightarrow{a} \langle nil, \mathtt{tt}\rangle$ and $\xrightarrow{b} \langle nil, \langle b\rangle_\emptyset\, \mathtt{tt}\rangle$. Nevertheless, only $\langle p, \psi_1\rangle$ is a successful node (see the definition of *test* at the end of this section).

We formally define the heuristic function $\widehat{h}$.

Definition 3.1 [$\widehat{h}(\langle p, \varphi\rangle)$] Let p be a CCS process, φ a SHM formula, ρ and $\mathcal{L}$ sets of visible actions, C a set of pairs $\{\langle x, \mathcal{L}'\rangle\}$, where x is a constant occurring in p, and $\mathcal{L}' \subseteq \mathcal{V}$, and $\mathcal{U}$ a set of constants. First, we define the auxiliary function $\widehat{h}$ with five arguments, $\widehat{h}(p, \mathcal{L}, \rho, C, \mathcal{U})$, inductively on p, as in Figure 3. Then, $\widehat{h}(\langle p, \varphi\rangle)$ is defined as $\widehat{h}(p, \emptyset, \rho, \emptyset, \emptyset)$, where ρ is the set of the first actions occurring on the formula φ [3].

The heuristic function $\widehat{h}(p, \mathcal{L}, \rho, C, \mathcal{U})$ is parametric with respect to a *restriction environment* $\mathcal{L}$, ($\mathcal{L} \subseteq \mathcal{V}$), keeping the set of actions on which some restriction holds. The function is initially applied to a process with $\mathcal{L} = \emptyset$ and $\mathcal{L}$ is modified when the function is applied to $p\backslash L$ (*Rule R5*), adding to $\mathcal{L}$ the actions in $L \cup \overline{L}$. Note that we expand the body of a constant x each time the environment under which that constant is evaluated has changed (*Rule R7*). Each constant already expanded is stored in C together with the current environment. Initially, $C = \emptyset$. The set ρ contains the first actions occurring on the formula under verification. Intuitively,

[3] More precisely, let φ and φ' be SHM formulae. The set of the first actions occurring in φ is inductively defined as follows:

$$\begin{array}{lllll} \mathcal{F}(\mathtt{tt}) & = & \mathcal{F}(\mathtt{ff}) & = & \emptyset \\ \mathcal{F}(\langle K\rangle_R\, \varphi) & = & \mathcal{F}([K]_R\, \varphi) & = & K \\ \mathcal{F}(\varphi \vee \varphi') & = & \mathcal{F}(\varphi \wedge \varphi') & = & \mathcal{F}(\varphi) \cup \mathcal{F}(\varphi') \end{array}$$

R1. $\widehat{h}(nil, \mathcal{L}, \rho, C, \mathcal{U}) = \infty$

R2. $\widehat{h}(\alpha.p, \mathcal{L}, \rho, C, \mathcal{U}) = \begin{cases} 0 & \text{if } \alpha \in \rho \cup \mathcal{L} \\ 1 + \widehat{h}(p, \mathcal{L}, \rho, C, \mathcal{U}) & \text{otherwise} \end{cases}$

R3. $\widehat{h}(p_1 + p_2, \mathcal{L}, \rho, C, \mathcal{U}) = \min(\widehat{h}(p_1, \mathcal{L}, \rho, C, \mathcal{U}), \widehat{h}(p_2, \mathcal{L}, \rho, C, \mathcal{U}))$

R4. $\widehat{h}(p_1|\cdots|p_n, \mathcal{L}, \rho, C, \mathcal{U}) =$

$$\begin{cases} \widehat{h}(\mathit{Unfold}^x(p_1|\cdots|p_n), \mathcal{L}, \rho, C, \mathcal{U} \cup \{x\}) \\ \quad \text{if } \exists \text{ an unguarded constant } x \text{ in } p_1|\cdots|p_n \text{ with } x \notin \mathcal{U} \\ 0 \\ \quad \text{if } p_i \text{ is guarded, } \ i \in [1..n], \text{ and } \exists i \in [1..n] \text{ s.t. } p_i = \alpha.q, \\ \quad \text{and } \alpha \in \rho \\ 1 + \widehat{h}(p_1|\cdots|q_k|\cdots|p_n, \mathcal{L}, \rho, C, \mathcal{U}) \\ \quad \text{if } p_i \text{ is guarded, } \ i \in [1..n], \text{ and } \exists k = \min\{j | p_j = \alpha_j.q_j, \ \alpha_j \notin \mathcal{L} \cup \rho\} \\ 1 + \widehat{h}(p_1|\cdots|q|\cdots|r|\cdots|p_n, \mathcal{L}, \rho, C, \mathcal{U}) \\ \quad \text{if } p_i \text{ is guarded and } \mathcal{F}\mathit{irst}(p_i) \subseteq \mathcal{L}, \forall i \in [1..n] \text{ and} \\ \quad \exists!\ \alpha \in \mathcal{L} \text{ s.t. } p_i = \alpha.q, p_j = \overline{\alpha}.r \ \text{ and } \alpha, \overline{\alpha} \notin \mathcal{F}\mathit{irst}(p_k) \\ \quad \forall k \in [1..n] \ k \neq i \neq j \\ D \\ \quad \text{otherwise} \end{cases}$$

where $D = \begin{cases} \infty & \text{if } \widehat{h}(p_i, \mathcal{L}, \rho, C, \mathcal{U}) = \infty, \forall i \\ \displaystyle\sum_{\substack{i=1, \\ \widehat{h}(p_i, \mathcal{L}, \rho, C, \mathcal{U}) \neq \infty}}^{n} \widehat{h}(p_i, \mathcal{L}, \rho, C, \mathcal{U}) & \text{otherwise} \end{cases}$

R5. $\widehat{h}(p\backslash L, \mathcal{L}, \rho, C, \mathcal{U}) = \widehat{h}(p, \mathcal{L} \cup L \cup \overline{L}, \rho, C, \mathcal{U})$

R6. $\widehat{h}(p[f], \mathcal{L}, \rho, C, \mathcal{U}) = \widehat{h}(p, f^{-1}(\mathcal{L}), f^{-1}(\rho), C, \mathcal{U})$

R7. $\widehat{h}(x, \mathcal{L}, \rho, C, \mathcal{U}) = \begin{cases} \infty & \text{if } \langle x, \mathcal{L} \rangle \in C \\ \widehat{h}(p, \mathcal{L}, \rho, C \cup \{\langle x, \mathcal{L} \rangle\}, \mathcal{U})) & \text{if } \langle x, \mathcal{L} \rangle \notin C \text{ and } x \stackrel{\text{def}}{=} p \end{cases}$

Fig. 3. The $\widehat{h}$ function for SHM formulae verification.

$\widehat{h}(\langle p, \varphi \rangle)$ returns the minimum number of actions to perform before performing an action occurring in the first modal operator in φ, i.e., an action in ρ. Nodes that

can perform actions in ρ are more promising. For $p = nil$ (*Rule R1*) the function $\widehat{h}$ returns ∞ since nil cannot perform actions in ρ as it is not able to perform any action at all. When applied to $\alpha.p$ (*Rule R2*), the function returns 0 if α is the first action of the formula (i.e., $\alpha \in \rho$) or if $\alpha \in \mathcal{L}$ (since the action α could be performed and so we optimistically return 0), otherwise the function is recursively applied for finding, if any, an action in ρ.

When the choice of two processes is encountered (*Rule R3*), the minimum number of actions before performing an action in ρ between the two components is returned. Now, let us consider the parallel composition of processes (*Rule R4*). In this case, the level of accuracy of the heuristic may vary according whether the requirement for it to be consistent is, or not, relaxed. In this definition we choose a syntactic-based method that, without considering all possible threads interleaving, simply returns a (perhaps not optimal) number of non-communication actions, not belonging to ρ, which could be performed by the threads. So, first we unfold once the unguarded constants occurring in the parallel composition. This can be done storing the unfolded constants in $\mathcal{U}$. Moreover, in the case where all the parallel components are guarded, if there exists a component of the parallel composition that can perform the first action of the formula, i.e., an action in ρ, then 0 is returned, as the process can perform immediately that action. Furthermore, if:

- there exists an independent component of the parallel composition, i.e., a process that can perform a non-restricted action not belonging to ρ ($p_i = \alpha.q$ and $\alpha \notin \mathcal{L} \cup \rho$); or
- all components can perform only restricted actions and there exists one and only one pair of processes that can communicate on a restricted action and this unique pair has the form $(\alpha.q, \overline{\alpha}.r)$;

then the estimated number of actions is 1 plus the value returned by a recursive application of the function.

In all the other cases, if the $\widehat{h}$-value of each process of the parallel composition is ∞, that is, each of them is not able to perform an action in ρ, so their composition and therefore we return ∞; if not, the sum of the number of actions by each parallel process p_i is returned, without considering the processes with infinite $\widehat{h}$-values. When considering a relabelled process (*Rule R6*), one must take as a set of first actions, the set $f^{-1}(\rho)$, and as a set of restricted actions, the set $f^{-1}(\mathcal{L})$ [4]. Finally, (*Rule R7*), ∞ is returned when we encounter a constant already expanded under the same environment (more precisely, when we encounter a constant x such that $\langle x, \mathcal{L}\rangle \in C$). In this case, no action in ρ has been found.

We explain *Rule R7* through a simple example. Consider $x \stackrel{\text{def}}{=} a.y$ and $y \stackrel{\text{def}}{=} b.x$. Suppose that we want to verify whether x satisfies $\varphi = \langle c\rangle_{\emptyset}\,\mathtt{tt}$. We apply our definition of $\widehat{h}$ with $\rho = \{c\}$. Therefore:

$$\widehat{h}(x, \emptyset, \rho, \emptyset, \emptyset) = \widehat{h}(a.y, \emptyset, \rho, \{\langle x, \emptyset\rangle\}, \emptyset) \text{ (by } \textit{Rule R7}, \langle x, \emptyset\rangle \notin C)$$

[4] Given a set of actions S, $f^{-1}(S) = \{\alpha \mid f(\alpha) \in S\}$.

$= \widehat{h}(y, \emptyset, \rho, \{\langle x, \emptyset\rangle\}, \emptyset)$ (by *Rule R2*)
$= \infty$ (by *Rule R7*, $\langle x, \emptyset\rangle \in C$).

The value ∞ means that the process x is not able to perform an action c.

We now explain *Rule R4*. Consider $x \stackrel{\text{def}}{=} a.x$, $y \stackrel{\text{def}}{=} \overline{a}.k_1$,$z \stackrel{\text{def}}{=} \overline{a}.k_2$ and $w \stackrel{\text{def}}{=} d.w$. Suppose that we want to verify whether $p = (x|y|z|w)\backslash\{a\}$ satisfies $\varphi = \langle c\rangle_\emptyset\, \mathtt{tt}$. We apply our definition of $\widehat{h}$ with $\rho = \{c\}$. Therefore:

$\widehat{h}(p, \emptyset, \rho, \emptyset, \emptyset) = \widehat{h}(x|y|z|w, \{a, \overline{a}\}, \rho, \emptyset, \emptyset)$ (by *Rule R5*)
$= \widehat{h}(a.x|\overline{a}.k_1|\overline{a}.k_2|d.w, \{a, \overline{a}\}, \rho, \emptyset, \{x, y, z, w\})$ (by*Rule R4 branch 1* four times)
$= 0$ (by *Rule R4 branch 5*).
Note that $\widehat{h}(d.w, \{a, \overline{a}\}, \rho, \emptyset, \{x, y, z, w\}) = \infty$

Once defined a heuristic function we can apply a search strategy. The search strategy expands nodes until it can deduce that the start node is a successful node or an unsuccessful one, based on the following function *test* that applied to a node n returns s if n is a successful node and u if n is an unsuccessful one:

- $test(\langle p, \mathtt{tt}\rangle) = s$;
- $test(\langle p, \varphi_1 \wedge \varphi_2\rangle) = s$ if $test(\langle p, \varphi_1\rangle) = s \wedge test(\langle p, \varphi_2\rangle) = s$;
- $test(\langle p, \varphi_1 \vee \varphi_2\rangle) = s$ if $test(\langle p, \varphi_1\rangle) = s \vee test(\langle p, \varphi_2\rangle) = s$;
- $test(\langle p, [K]_R\, \varphi\rangle) = s$ if either $\widehat{h}(\langle p, [K]_R\, \varphi\rangle) = \infty$ or $test(m) = s\ \forall m \in \{n' | n \xrightarrow{\alpha} n',\ \alpha \notin R\}$;
- $test(\langle p, \langle K\rangle_R\, \varphi\rangle) = s$ if $\exists m \in \{n' | n \xrightarrow{\alpha} n',\ \alpha \notin R\}$ such that $test(m) = s$.
- $test(p, \mathtt{ff}) = u$;
- $test(\langle p, \varphi_1 \wedge \varphi_2\rangle) = u$ if $test(\langle p, \varphi_1\rangle) = u \vee test(\langle p, \varphi_2\rangle) = u$;
- $test(\langle p, \varphi_1 \vee \varphi_2\rangle) = u$ if $test(\langle p, \varphi_1\rangle) = u \wedge test(\langle p, \varphi_2\rangle) = u$;
- $test(\langle p, \langle K\rangle_R\, \varphi\rangle) = u$ if either $\widehat{h}(\langle p, \langle K\rangle_R\, \varphi\rangle)) = \infty$ or $test(m) = u\ \forall m \in \{n' | n \xrightarrow{\alpha} n',\ \alpha \notin R\}$;
- $test(\langle p, [K]_R\, \varphi\rangle) = u$ if $\exists m \in \{n' | n \xrightarrow{\alpha} n',\ \alpha \notin R\}$ such that $test(m) = u$.

Consider now the simple CCS process $p = a.b.g.x + a.d.b.x + b.d.b.c.nil$; with $x \stackrel{\text{def}}{=} b.d.nil$. The transition system for p has 11 states. Let us suppose that we want to verify the following SHM formula:

$\varphi \stackrel{\text{def}}{=} [a]_\emptyset\, \langle b\rangle_\emptyset\, \mathtt{tt}$: "after each action a an action b can be performed".

It holds that p satisfies φ. We note that for φ, visiting only 7 nodes is sufficient in order to state that the formula is satisfied for the process p, while the standard transition system for p has 11 states.

Let us apply our approach to the process p with the Greedy search strategy. The node expansion terminates when enough nodes are selected to deduce whether the start node is a successful node or an unsuccessful one, based on the definition of the function *test*.

In Figure 4 the $\widehat{h}$-value of each node is reported. For instance, it holds that

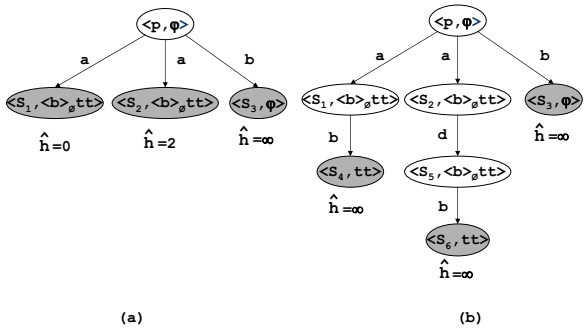

Fig. 4. A simple example with Greedy and $\widehat{h}$.

$\widehat{h}(\langle S_1\,^{5}, \langle b\rangle_{\emptyset}\,\mathtt{tt}\rangle) = 0$ meaning that process S_1 can immediately perform the first action of the corresponding formula, that is b. Applying the Greedy search strategy, first we expand the node $\langle p, \varphi\rangle$: three states are generated: $\langle S_1, \langle b\rangle_{\emptyset}\,\mathtt{tt}\rangle$, $\langle S_2, \langle b\rangle_{\emptyset}\,\mathtt{tt}\rangle$ and $\langle S_3, \varphi\rangle$ (see Figure 4(a)). In both figures, 4(a) and 4(b) the shaded nodes represent the states to be expanded. Then, we choose to expand first $\langle S_1, \langle b\rangle_{\emptyset}\,\mathtt{tt}\rangle$ that has the least h-value (equal to 0), generating $\langle S_4, \mathtt{tt}\rangle$, and then $\langle S_2, \langle b\rangle_{\emptyset}\,\mathtt{tt}\rangle$, whose h-value is equal to 1. Finally, we expand $\langle S_5, \langle b\rangle_{\emptyset}\,\mathtt{tt}\rangle$, with h-value equal to 0, generating $\langle S_6, \mathtt{tt}\rangle$ (see Figure 4(b)). Based on the definition of successful/unsuccessful node, since $\langle S_4, \mathtt{tt}\rangle$, $\langle S_6, \mathtt{tt}\rangle$ and $\langle S_3, \varphi\rangle$ are successful nodes, we can stop the construction of the transition system stating that the process p satisfies the formula φ.

4 Experimental Results

The proposed approach has been implemented in a tool that is an extension of DELFIN^{+} (DEadloack FINder) [18,19]. In fact, the previous version of the tool was specific for deadlock detection, providing a set of heuristics to be used with informed search strategies like A*, IDA* and Greedy. The added functionality includes the possibility to verify CCS processes against any kind of formula expressed in the selective Hennessy-Milner logic, by applying the formula-based heuristic presented in Section 3.

We run some experiments on an Intel Pentium 4 with a 2.80 GHz processor and 1 GB of RAM, and, in general, could observe encouraging results. We selected from the literature a sample of well known systems and some interesting properties to be verified, which are described below.

Multicast Protocol for Mobile Computing (MPMC):a protocol for reliable multicast in distributed mobile systems, presented in [2,3]. On this system we checked the following two SHM formulae:

$$\varphi_1 = [\overline{deliver(m, msg)}]_{\{send(msg)\}}\,\mathtt{ff}$$

[5] $S_1 = b.g.x.$

expressing the property that any multicast delivered by a group member has been originated by a group member; and

$$\varphi_2 = [\overline{deliver(m, msg)}]_\emptyset \, [\overline{deliver(m, msg)}]_\emptyset \, \mathtt{ff}$$

expressing the property that no group member delivers duplicate multicasts, i.e. duplicates are discarded. Both properties are true and the size of the labelled transition systems constructed by the Concurrency Workbench of the New Century (CWB-NC) [11] consists of $4,815$ states. With our tool, property φ_1 was verified after 146 generated states while for property φ_2 no reduction was obtained.

GRID: two processes on a grid 5×5 of relay stations which allow them to communicate. This example is taken from [6]. On this system we considered the trivial SHM formula $\varphi_1 =< connected >_\emptyset \, \mathtt{tt}$, expressing the property that there exists a path to connect the two processes. The property is satisfied. The CWB-NC fails to verify any formula on that process as its labelled transition system is too big to be loaded. With our tool, we verified the formula with two processes initially positioned at $(1, 1)$ and $(5, 5)$ respectively, and the true result was given after $11,509$ generated states.

MUTUAL: a system handling the requests of a resource shared by 10 processes. It presents two alternative choices between a server based on a *round robin scheduling* and a server based on *mutual exclusion*. On this system we checked the following two SHM formulae

$$\varphi_1 = [\overline{roundrobin}]_\emptyset \, [\overline{work}_{i+1}]_{\{\overline{work}_i\}} \, \mathtt{ff}$$

$$\varphi_2 = [\overline{work}_{i+1}]_{\{\overline{work}_i\}} \, \mathtt{ff}$$

φ_1 expresses the property that when a server based on a round robin scheduling is chosen, process $i + 1$ cannot use the resource before process i. This property is satisfied, instead the system does not satisfy φ_2 expressing the property that, in general, process $i+1$ cannot use the resource before process i. The CWB-NC builds a system consisting of $32,768$ states, while with our tool, with $i = 1$, we managed to verify formula φ_1 after $13,824$ generated states and formula φ_2 after 89 states.

Philips Bounded Retransmission Protocol (BRP): the Bounded Retransmission Protocol used by the Philips Company in one of its products [17,21,22]. On this system we checked the following two SHM formulae:

$$\varphi_1 = [\overline{in(d1\_di)}]_\emptyset \, [\overline{in(ok)}]_{\{\overline{out(di,ok)}\}} \, \mathtt{ff}$$

$$\varphi_2 = [\overline{in(d1\_di)}]_\emptyset \, \langle \overline{in(nok)} \rangle_\emptyset \, \mathtt{tt}$$

The formula φ_1 means that, after accepting a data packet of length i, the protocol cannot issue an ok confirmation (action $\overline{in(ok)}$) to the sending client unless the last segment of the packet (action $\overline{out(di, ok)}$) has been delivered to the receiving client. The formula φ_2 means that, after accepting a data packet of length i, the protocol can issue a nok confirmation (action $\overline{in(nok)}$) to the sending client. Both properties are satisfied. The size of the labelled transition system computed by the CWB-NC

is of 759 states, while with our tool, with $i = 3$, we could verify φ_1 after 558 states and φ_2 after 34 states.

Solid State Interlocking (SSI): the British Rail's Solid State Interlocking (SSI) whose specification is given in [7]. This system is devoted "to adjust, at the request of the signal operator, the setting of signal and points in the railway to permit the safe passage of trains." On this system we checked the folowing SHM formula $\varphi_1 = [det]_{\{fail\}}\,\mathtt{ff}$. The formula means that a failure is detected only if it has actually occurred. The system does not satisfy φ_1. For that process, the CWB-NC constructs a labelled transition system of $3,616$ states while with our tool we could verify the formula to be false after generation of 361 states.

Solitaire Game (SG): a CCS specification of a solitaire game [4], developed by Luca Aceto (available at [1]). This system satisfies the following SHM formula: $\varphi_1 = \langle \overline{w1}, \overline{w2}, \overline{w3}, \overline{b5}, \overline{b6}, \overline{b7} \rangle_{\emptyset}\,\mathtt{tt}$. The labelled transition systems of this process by the CWB-NC consists of $3,107$ states but our tool employed 431 states to verify φ_1.

In addition to those processes, for a better evaluation of our heuristic we considered a deadlocked version of the dining philosophers process and checked the possibility for one specific philosopher to eat. We did this on several instances of that process, i.e., by increasing the number of philosophers. The results of all runs are reported in Table 1: n is the number of philosophers and the figures indicate the number of generated states to verify the formula in the corresponding run, both by our tool with Greedy search and by the CWB-NC.

n	5	7	9	11	13	15	20	.	30
Greedy	177	440	816	1389	2055	2847	5387	.	12867
CWB-NC	2404	54142	–	–	–	–	–	–	–

Table 1
Results on the dining philosophers

As one can see from the table, the CWB-NC is not able to load the process from $n = 9$ on, while with our tool we managed to model check even a configuration of 30 philosophers.

5 Related work and Conclusion

In this paper we presented a method and a tool for directed model checking applied to concurrent systems described as CCS processes. The properties (both weak-liveness and safety) to be verified are expressed in the Selective Hennessy-Milner logic [5], which is a branching-time logic. The efficiency of the proposed approach is shown on some examples, well known in literature, where the Greedy strategy has been applied together with the heuristic of Figure 3.

The work most closely related to this is presented in [12] where the authors propose a heuristic approach based on the structure of the LTL (Linear Temporal Logic) properties to be verified. However, their work focuses on liveness properties

and invariant and assertion errors. The approach in [12] is further refined with Bayesian reasoning in [31] where the focus is not the study and the definition of heuristics but the application of statistical methods to improve the expected behavior of guided search. This is obtained by interpreting heuristic estimates as random variables rather than point values. In [13], a Hamming Distance heuristic has been used to guide the error search in SPIN. This technique requires that a specific state of the system, in which an error occurs, is identified initially; the Hamming Distance between that state and the current system state is then used to guide the search for a short counterexample that reaches the given violation state. In [24] heuristics have been applied to the validation of communication protocols. Some kinds of safety properties, like absence of deadlock, are considered.

In a precursor paper [30] one of the authors combines AO* [27] with local model checking to verify CCS processes. This was the first initiative to verification by directed model checking, providing an example of heuristic function. As an extension to that work, we presented another heuristic function definition that can be easily tailored to deadlock detection. In fact, in [19] the authors propose deadlock-specific heuristics and their efficiency is analyzed on several examples through DELFIN$^+$. Thus, once defined the heuristic function, and based on the formula, several heuristic search strategies can be used (Greedy, A*, IDA*, AO* etc.). Finally, other well-known reduction techniques, such as partial order, compositional reasoning, abstraction, can be applied as an alternative to local model checking.

As a future work we intend to proceed with analyzing more accurate heuristic functions and to deepen the evaluation of the method on other case studies. We are interested in combining this approach with abstraction techniques, in order to achieve further improvements.

References

[1] L. Aceto. *The Solitaire.* http://www.cs.auc.dk/~luca/DAT4/solitaire.cwb

[2] G. Anastasi, A. Bartoli, N. De Francesco. *Efficient Verification of a Multicast Protocol for Mobile Computing.* The Computer Journal, 44(1), (2001). 21-30.

[3] G. Anastasi, F. Spadoni, A. Bartoli. *Group Multicast in Distributed Mobile Systems with Unreliable Wireless Network.* In *SRDS '99: Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems.* IEEE Computer Society, 14, (1999). 14-23.

[4] A. Arnold, D. Begay, P. Crubille. *Construction and analysis of transition systems with MEC.* World Scientific, Chapter 6, 1994.

[5] R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. Selective mu-calculus and Formula-Based Abstractions of Transition Systems. *Journal of Computer and System Sciences, 59(3)*, 1999, 537-556.

[6] J. Bradfield, C. Stirling. *Verifying Temporal Properties of Processes.* In *CONCUR '90: Theories of Concurrency - Unification and Extension (Proc.).* Springer, 1990. 115-125.

[7] G. Bruns. *A Case Study in Safety-Critical Design.* In *CAV '92: Proceedings of the Fourth International Workshop on Computer Aided Verification* Springer, 1992. 220-233.

[8] E.M. Clarke, O. Grumberg, D.E. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems, 16(5)*, 1994. 1512-1542.

[9] E.M. Clarke, O. Grumberg, D. Peled. Model Checking. *MIT press*, 2000.

[10] E.M. Clarke, D.E. Long, K.L. McMillan. Compositional Model Checking. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science*, 1989. 353-362.

[11] R. Cleaveland, S. Sims. The NCSU Concurrency Workbench. In *Proceedings of the Eighth International Conference on Computer-Aided Verification (CAV'96)*, Lecture Notes in Computer Science 1102, 1996. 394–397.

[12] S. Edelkamp, A. Lluch-Lafuente, S. Leue. Directed Explicit Model Checking with HSF-SPIN. In *Proceedings of the 8th International SPIN Workshop on Model Checking Software*, Lecture Notes in Computer Science 2057, 2001. 57-79.

[13] S. Edelkamp, A. Lluch-Lafuente, S. Leue. Trail-directed model checking. *Electronic Notes in Theoretical Computer Science (ENTCS)* 55, 2001.

[14] S. Edelkamp, A. Lluch-Lafuente, S. Leue. Directed explicit-state model checking in the validation of communication protocols. *Int J Softw Tools Technol Transfer 6*, 2004. 257-259.

[15] P. Godefroid. Partial-Order Methods for the Verification of Concurrent Systems. LNCS 1032, 1996.

[16] P. Godefroid, S. Khurshid. Exploring Very Large State Spaces Using Genetic Algorithms. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'02)*, Lecture Notes in Computer Science 2280, 2002. 266-280.

[17] J. F. Groote, J. van de Pol. *A Bounded Retransmission Protocol for Large Data Packets.* Algebraic Methodology and Software Technology, 1996. 536-550.

[18] S. Gradara, A. Santone, M.L.Villani. Using Heuristic Search for Finding Deadlocks in Concurrent Systems. *Information and Computation, 202(2)*, 2005. 191-226.

[19] S. Gradara, A. Santone, M.L.Villani. *DELFIN+: An efficient deadlock detection tool for CCS processes.* to appear in *Journal of Computer and System Sciences.*

[20] S. Gradara, A. Santone, M.L.Villani. *Directed Model Checking CCS processes.* Technical Report, TR-RCOST 12/03, February 2006.

[21] K. Havelund, N. Shankar. *Experiments in Theorem Proving and Model Checking for Protocol Verification.* In *FME '96: Industrial Benefit and Advances in Formal Methods*, Springer-Verlag, 1996. 662-681.

[22] L. Helmink, M. P. A. Sellink, F. W. Vaandrager. *Proof-checking a data link protocol.* In *TYPES '93: Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science, 806, 1994. 127-165.

[23] C. Jard, T. Jéron. Bounded-memory Algorithms for Verification on-the-fly. In *Proceedings of the Third International Conference on Computer-Aided Verification (CAV'91)*, LNCS 575, 1991. 192-201.

[24] F.J. Lin, P.M. Chu, M.T. Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. ACM, 1988. 126-135.

[25] K. McMillan. Symbolic Model Checking. *Boston: Kluwer Academic Publishers*, 1993.

[26] R. Milner. Communication and Concurrency. *Prentice-Hall*, 1989.

[27] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley.

[28] D. Peled. All from One, One for All, on Model-Checking Using Representatives. In *Proceedings of the Fifth International Conference on Computer-Aided Verification (CAV'93)*, LNCS 679, 1993. 409–423.

[29] A. Santone, *Automatic Verification of Concurrent Systems using a Formula-Based Compositional Approach*, Acta Informatica, **38(2)**, 2002, 531-564.

[30] A. Santone. *Heuristic Search + Local Model Checking in Selective mu-Calculus.* IEEE Transactions on Software Engineering, 29(6), 2003. 510-523.

[31] K.S. Seppi, M. Jones, P. Lamborn. Guided Model Checking with a Bayesian Meta-heuristic. *Fundamenta Informatica*, To appear, 2005.

[32] C. Stirling. An Introduction to Modal and Temporal Logics for CCS. In *Concurrency: Theory, Language, and Architecture*, Lecture Notes in Computer Science 391, 1989.

[33] C. Stirling, D. Walker. Local Model Checking in the Modal Mu-Calculus. TCS, 89, 1991. 161-177.

[34] C.H. Yang, D.L. Dill. Validation with Guided Search of the State Space. In *Proceedings of the 35th Conference on Design Automation (DAC'98)*, 1998. 599-604.